SANTIAGO CERIA

COMPUTACION
Algoritmos

Unidad 3

Cátedra de Computación

BIBLIOTECA DEL
CICLO BASICO

3





EUDEBA S.E.M.
Fundada por la Universidad de Buenos Aires

EDITORIAL UNIVERSITARIA DE BUENOS AIRES
Sociedad de Economía Mixta
Rivadavia 1571/73
Hecho el depósito que marca la ley 11.723
ISBN 950-23-0148-X
IMPRESO EN LA ARGENTINA

UNIDAD 3 - ALGORITMOS

3.0 OBJETIVOS

OBJETIVOS PRINCIPALES

a) Capacidad para encontrar y expresar con precisión soluciones algorítmicas de problemas simples, no triviales.

b) Habilidad en el uso del método de diseño por refinamientos sucesivos, dando siempre soluciones estructuradas.

c) Conocimientro de la existencia de herramientas para la demostración rigurosa de la correción de algoritmos y nociones intuitivas sobre su utilización.

OBJETIVOS SECUNDARIOS

d) Conocimiento de los elementos principales de un lenguaje de programación

e) Conocimiento de la existencia de técnicas para la definición y análisis de lenguajes formales.

f) Noción de la magnitud del problema de la construcción de algoritmos.

ALGORITMOS

3.1 Concepto de Algoritmo

Los métodos algorítmicos, aunque en forma inconciente, forman parte de nuestra manera de solucinar problemas. Alguna vez nos hemos referido al algoritmos de la extracción de la raiz cuadrada o al algoritmo de Euclides, al algoritmo de la suma decimal. Todos estos ejemplos que acabamos de mencionar son procedimientos encaminados a la solución de un conjunto de problemas de un mismo tipo.

Podemos decir que un algoritmo es una secuencia finita de instrucciones encaminada a lograr la solución de una familia de problemas, que satisface las siguientes condiciones:

- Termina, esto es, conduce a una secuencia finita de acciones.
- Da siempre la solución al problema a que se aplica.
- Cada instrucción es realizable, o sea, el procesador debe poder ejecutarla en un tiempo finito.
- No es ambigua, esto es, que cada una de las instrucciones en cada paso determinan inequívocamente la acción a ser tomada.

Se llaman procedimientos a las secuencias de instrucciones que cumplen los dos últimas condiciones impuestas a los algoritmos. Los procedimientos pueden por lo tanto no finalizar, y comprenden a los algoritmos.

Ej. .1 Diga cuales de los siguientes mecanismos aprendidos en la escuela son algoritmos o pueden transformarse en un algoritmo.

a) Mecanismo de la multiplicación.

b) Mecanismo para obtenér la representación decimal de una fracción.

c) Mecanismo para obtener la raíz cuadrada en los números reales, de un número entero.

d) Mecanismo de la división entera.

Hata el momento hemos hablado de algoritmos expresados en tres lenguajes distintos. Los algoritmos aprendidos en la escuela, que nos han sido expresados en castellano, los algoritmos que hemos desarrollado en el lenguaje de la máquina de Turing y los algoritmos expresados en Timba. La primera forma de descripción de algoritmos, el lenguaje castellano, resulta impreciso. El segundo, el de la máquina de Turing, es absolutamente preciso pero muy alejado de nuestra manera de pensar lo que hace prácticamente imposible su uso para el planteo de problemas complejos. El tercero, Timba, se adapta perfectamente para la resolución de problemas de un tipo muy específico, los únicos objetos que permite manejar son cartas y pilas, de él por lo tanto no es posible descubrir ningún procedimiento que requiera la manipulación de otros objetos, como por ejemplo el contar la cantidad de cartas de una pila, que requiere trabajar con números enteros.

En la próxima sección veremos un lenguaje y un método que nos permitirán abordar problemas complejos y llegar a soluciones expresadas con total precisión.

Nuestro interés se encuentra en esta etapa del curso en la solución de problemas mediante algoritmos. Si bien hasta el presente este método ha si-

do casi el camino obligado para la utilización de la computadora, no es el único posible y más adelante veremos otros enfoques que ya son utilizados en algunas aplicaciones actuales y que abren promisorias sendas para el futuro.

3.2 La búsqueda de una forma de expresar algoritmos

La alta velocidad de operación -millones de operaciones por segundo- y la gran confiabilidad de las computadoras actuales, ha dotado a la humanidad de una posibilidad hasta ahora inédita para encarar la resolución de problemas. Ella consiste en plantear algoritmos cuya ejecución puede insumir millones de pasos. Hasta el presente una solución de este tipo hubiera tenido, a los sumo interés teórico. En el presente puede ser no sólo factible, sino también económica. Los procesadores al ser cada vez más pequeños, más baratos y más veloces, amplian constantemente su zona de aplicación invadiendo todas las áreas tecnológicas. Es así que cada vez más nuestro entorno esta gobernado por algoritmos, desde el funcionamiento de un simple reloj de muñeca hasta el más sofisticado diagnóstico médico.

Esta enorme necesidad de implementar algoritmos mediante la programación de computadoras, que supone una inversión mundial de miles de millones de dólares anuales, ha trasformado el arte de la programación en la ingeniería del software.

En la primera etapa de esta evolución los esfuerzos se dirigieron hacia el problema de la síntaxis. Es decir, a la obtención de métodos de descripción formal de lenguajes de computación, y la construcción de los algoritmos capaces de traducirlos al lenguaje de máquina (Compiladores). Los logros obtenidos en el estudio de la sintaxis ha derivado la atención al problema de la semántica (estudio de símbolos respecto de su significado) este tema será desarrollado más adelante.

3.3 Pascal - CBC

El lenguaje que vamos a introducir esta basado en el lenguaje PASCAL, y lo llamaremos PASCAL-CBC.

PASCAL-CBC va a comprender las mismas sentecias de control de TIMBA- Selección (si-entonces, si-entonces-sino) con algunas diferencias en la delimitación de los alcances- Iteración (mientras). Va a manejar un repertorio más amplio de objetos y va a permitir realizar nuevas operaciones sobre ellos.

Por razones didácticas este lenguaje será introducido en dos etapas comprensivas 0 y 1.

La descripción detallada de ambos puede verse en el apéndice de cartas sintácticas de PASCAL-CBC.

3.3.1 Variables

PASCAL-CBC, como la mayoría de los lenguajes trabaja con variables. Para definir una variable debemos asociarle un nombre o identificados y un conjunto sobre el cual puede tomar valores.

El nombre de una variable sirve para identificarla, permaneciendo inalterable durante todo su campo de definición; mientras que su valor podrá ser modificado.

Haciendo uso del modelo del casillero utilizado para describir al funcionamiento de una computadora, podemos asimilar una variable a una casilla sobre la cual se coloca una etiqueta con su nombre. Podemos asimismo pensar que el casillero contiene varios tipos de casillas, destinadas a almacenar distintos objetos. En el momento de definir la variable, o sea, en este caso, al pegar la etiqueta con su nombre, deberemos pensar en los objetos que deberá contener para seleccionar el tipo de casilla adecuado, esto significa, determinar el conjunto sobre el cual va a tomar valores dicha variable.

Las variables podrán ser utilizadas en expresiones, así, si el valor de la variable A es el entero 2, A+(A*3) valdrá 8.

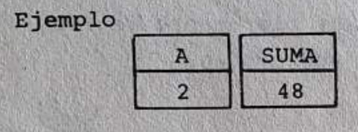
También sobre ellas podrá realizarse la operación de asignación, que en nuestro lenguaje tiene el siguiente formato

<variable>:= expresión

Esto se lee: "<variable> recibe <expresión>

Una vez ejecutada esta operación el valor de la variable pasará a ser el resultado de la evaluación de la expresión.

Ejemplo



SUMA :=SUMA · A * 4

	SUMA
	56

3.3.2 Expresiones

Las expresiones de PASCAL-CBC quedan definidas por la correspondiente carta sintáctica. Como podemos apreciar en ella, a las operaciones aritméticas habituales, se han agregado las las operaciones lógicas (∧,∨, ⌐), las relacionales (=, < , > , <= ; >=, <>) y la operación aritmética MOD, cuyo significado veremos a continuación.

3.3.2.1 operaciones relacionales

Las operaciones relacionales da valores de verdad

1 - verdadero- si la relación se cumple

0 - falso- si la relación no se cumple

Ejemplos:

1) 2=2 arroja como resultado 1, es decir, verdadero .

2) si x=9

x <= 9 vale 1

x < 9 vale 0

3.3.2.2 Operaciones lógicas

Las operaciones lógicas actúan sobre valores de verdad, dando como resultado nuevos valores de verdad.

La operación negación ' ⌐ ', que se lee 'no', actúa sobre un único valor

de verdad, cambiándolo.

La operación conjunción '∧', que se lee 'y', da verdadero si sus dos operandos son verdaderos, en cualquier otro caso da falso.

Esto puede sintetizarse en las siguientes 'tablas de verdad'

¬	
1	0
0	1

∧	0	1
0	0	0
1	0	1

∨	0	1
0	0	1
1	1	1

→ '∨' operación disyunción, se lee 'o'

Por último el operador MOD se define como el resto de la división entera de dos números. Es decir, si a=10, y b=3 a mod b da como resultado 1.

Asimismo asumimos la existencia de las siguientes funciones preestablecidas.

rcuad (n)	Devuelve la raíz cuadrada de n
abs (n)	Devuelve el valor absoluto de n

Ej. 3.2 Evalúe las siguientes expresiones asumiendo que a=2, b=5 y c=23

1) a <=3 ∧ b > c
2) a > b
3) ((a+3 >= b) ∧ (1 <= a ∨ c >= a + b))
4) abs (-a)

Nuestro lenguaje utiliza un conjunto de palabras preestablecidas que llamaremos palabras reservadas, y que para distinguirlos claramente de los símbolos definidos por el programados escribiremos subrayadas.

3.3.3 Estructura de un programa PASCAL-CBC-0

Un programa PASCAL-CBC-0 estará constituido por dos partes; en la primera se definen las variables mientras que en la segunda se especifican las acciones a efectuarse mediante las sentecias correspondientes. Esta segunda parte comenzará con la palabra reservada 'comienzo' y terminará con la palabra reservada 'fin'.

3.3.3.1 definición de variables

La definición de variables se realizará mediante la palabra reservada VAR seguida de la lista de definiciones, separadas por ';', cada una de estas definiciones esta constituida por el identificador de la variable seguido de ':' y su tipo, esto es, el conjunto sobre el cual toma valores.

En PASCAL-CBC-0 los tipos permitidos son Entero y Booleano. En el primer caso el conjunto es el de los números enteros. Una variable booleana tomará valores lógicos (verdadero y falso).

Pueden agruparse definiciones del mismo tipo escribiendo los identificadores correspondientes separados por comas y a continuación ':' y el tipo.

Ejemplo:

```
VAR
    i: entero; sueldo:entero;
    lados:entero; cierto:booleano;
```

Otra forma de definir lo mismo es:

```
VAR
    i,sueldo,lados:entero;
    cierto:booleano;
```

3.3.3.2 Sentencias

ASIGNACION

Es la operación de asignación de la que ya hemos hablado

Antes de seguir veamos un ejemplo: El siguiente es un programa que deja en Z la suma de los cinco primeros números impares

```
(3.2)       var    Z : entero ;
                   X : entero
            comienzo      Z:= 0 ; X:= 1 ;
                          Z:= Z + X ; X:= X + 2 ;
                          Z:= Z + X ; X:= X + 2 ;
                          Z:= Z + X ; X:= X + 2 ;
                          Z:= Z + X ; X:= X + 2 ; Z:= Z + X
            fin
```

```
(3.3)   var
            A,B,C: booleano ;
        comienzo
            A:= verdadero ; B:= falso
            C:= A∧B  (C recibirá el valor booleano falso)
            A:= ¬A∨C (A recibirá el valor booleano falso)
        fin
```

Ej. 2 Determine los valores de A,B y C al terminar el siguiente programa

```
        var
            A,B,C: booleano ;
        comienzo
            A:= verdadero ; B:= falso ; C:= verdadero
            A:= ¬A∧B
        B:= A∧B∧C
        C:= ¬(A∧B∨C)∧B
        fin
```

SELECCION

Cada sentencia lleva, ya sea implicitamente o explícitamente, información sobre cual será la próxima sentencia a ejecutarse. Es fácil ver que la asignación no debe alterar la ejecución en secuencia del programa, es decir, la próxima sentencia a ejecutarse será la que siga en orden de escritura. Pero si intentamos especificar algoritmos exclusivamente con este tipo de estructuras de control, nos vamos a encontrar conque sólo podemos expresar una cantidad limitada de ellos.

La selección es una estructura que nos permite alterar esta secuencia de ejecución, de modo tal que podamos decidir cual camino tomar segun que

una condición se cumpla o no.

La forma de esta sentencia es:

si <expresión> entonces <sentencia 1> sino <sentencia 2>

Es decir, si la <expresión> es verdadera, entonces ejecutará <sentencia 1>, si la <expresión> es falsa, ejecutará <sentencia 2>.

En ambos casos la ejecución continúa con la sentencia siguiente.

Una seguna forma de esta sentencia es:

si <expresión> entonces <sentencia>

La segunda forma puede considerarse como una abreviatura de la primera en la que <sentencia 2>es nula

REPETICION

Es frecuente encontrarse con problemas cuya resolución conduce a repetir un conjunto de acciones según una determinada condición, las sentencias que para esto provee PASCA-CBC-0 son:

mientras <expresión> hacer <sentencia>

Al ejecutarse esta sentencia, mientras <expresión>sea verdadera se ejecutará <sentencia> repetidas veces, cuando la <expresión> sea falsa, seguirá en secuencia.

repetir <sentencia> hasta <expresión>

Se ejecutará la <sentencia> hasta que la <expresión> sea verdadera.

Es fácil ver que ambos casos, para que las repeticiones no sean infinitas, la <sentencia> deberá alterar la <expresión>

EJEMPLOS

Reescribamos el ejemplo (2.2)

```
var  z,i :entero  ; x :=entero
comienzo
     i:= o ;  x:=1  ; z:=0;
     mientras  i< 5  hacer
                    comienzo
                       z:= z+x  ; x:=x+2 ;
                       i:= i+1
                    fin
fin
```

Que también puede escribirse:

```
var  z,i: entero ; x : entero
comienzo
     i:= 0 ;     z:=0 ; x:=1
     repetir
         z:= z+x; x:=x+2 ;
```

```
        i:= i+1
    hasta  i <= 4
fin
```

3.3.3.3 Comentarios

En cualquier lugar de un programa se pueden insertar comentarios encerrados entre llaves, los mismos sirven exclusivamente para clarificar la acción del programa y son ignorados por el procesador.

3.3.3.4 Procedimientos preestablecidos

Para terminar con nuestro PASCAL-CBC-0 debemos incluir dos procedimientos preestablecidos:

Leer (<variable>) ⟶ lee un campo de la unidad de entrada

Imprimir (<expresión>) ⟶ presenta el resultado de evaluar la <expresión> en la unidad de salida

3.4 Refinamiento por pasos sucesivos

Para llegar a escribir un programa en lenguaje de computación que resuelva un determinado problema avanzaremos por pasos sucesivos desde una planteo global inicial, agregando en cada paso una mayor cantidad de detalles. De esta manera en cada paso podremos focalizar nuestra atención en unos pocos aspe os y asegurarnos de que la solución que damos realmente los resuelva.

En cada uno e estos pasos trabajaremos con un esquema de programa en el que podrán fig ar como sentencias o expresiones, descripciones castellanas de procesos. Cada paso será un refinamiento del anterior, esto es, se obtendrá por el desgloce de alguna de esas especificaciones.

Ejemplos

Imprimir la suma de los números impares menor o iguales que 2000

Un primer paso sería

```
comienzo
      sumar los primeros 2000 números impares;
      imprimir la suma;
fin
```

Tiene sentido comenzar con un paso tan trivial dado que si existiera una máquina capaz de interpretarlo nuestro problema estaría resuelto, Como PASCAL-CBC- no brinda esta facilidad, debemos seguir refinando.

```
var
      n:= entero ;
comienzo
      n:= primer impar
      mientras n<=2000 hacer
      comienzo
```

```
    comienzo
        sumar n
        n:= próximo impar
    fin
    imprimir (la suma)
fin
```

Para poder refinar 'sumar n' necesitamos de una variable que inicialmente tenga el valor 0 a la que se suma n en cada paso de la iteración.

```
var
    n,suma: entero;
comienzo
    suma:=0;
    n:= primer impar;
    mientras  n<=2000 hacer
        comienzo
            suma:= suma + n;
            n:= próximo impar;
        fin
    imprimir(suma)
fin
```

Teniendo en cuenta que el primer impar es 1 y cada uno de los demás se obtiene sumando 2 al anterior, llegamos a la versión final de nuestro programa:

```
var
    n,suma: enero;
comienzo
    suma:=0;
    n:=1
    mientras  n<= 2000 hacer
        comienzo
            suma:= suma +n;
            n:= n+2;
        fin
    imprimir(suma)
fin
```

Veamos otro ejemplo más complicado:

Tratemos de realizar un programa que dado un número imprima sus factores primos. Una primera aproximación puede ser la siguiente:

```
var
    n:enero;
```

```
comienzo
     leer (n);
  (a) Imprimir los factores primos de n:
fin
```

Para refinar (a) deberemos recorrer todos los posibles candidatos a factores primos de n

```
var
     p,n:entero;
comienzo
     leer (n);
  (c) p:= primer primo;
     mientras p<= n  hacer
     comienzo
      (b) si p es factor primo de n,imprimirlo
          p:= p+1
     fin
fin
```

Refinemos (b) y (c)

```
var
     p,n:entero;
comienzo
     leer (n);
     p:=2;
     mientras p<=n  hacer
     comienzo
         si p divide a n entonces
        (d)   si p es primo imprimirlo
         p:= p+1
       fin
fin
```

Refinando (d) llegamos a:

```
(33) var
          p,n: entero;
       primo:booleano;
   comienzo
       leer (n);
       p:= 2;
```

```
mientras p<=n hacer
comienzo
    si p divide a n entonces
        comienzo
            primo:= (p es primo);
            si primo entonces imprimir (p);
        fin
    p:= p+1
fin
fin
```

Aqui estamos usando la variable lógica o booleana 'primo' cuyo valor deberá ser verdadero si p es primo y falso en caso contrario. Quedan dos problemas a resolver:

a) determinar si p divide a n, lo que se resulve teniendo en cuenta que ello es equivalente a que el resto de la división de n por p sea cero o sea que n mod p =0.

b) Resolver la asignación

   primo:= (p es primo)

   Para determinar la verdad de esta última relación podemos ver si p es divisible por algún número menos que él. Esto puede realizarse con el siguiente segmento de programa:

(3.4)
```
primo:= verdad
i=2;
mientras (i< p)∧ primo hacer
    comienzo
        primo:= (p mod i)<> 0 ;
        i:= i+1
    fin
```

Este ciclo se realizará hasta que deja de cumplirse alguna de las dos condiciones de la sentencia mientras, o sea hasta que:

i) La variable primo sea falsa, lo cual significa que se ha hallado i tal que p mod i = 0 que equivale a decir que i divide a p, en cuyo caso es correcto terminar el ciclo y dejar ese valor de 'primo' porque p no es primo.

ii) i haya alcanzado el valor p sin que ninguno de los números menores que él lo hayan divido exactamente, debiendo entonces dejarse en 'primo' el valor verdadero que posee, porque p es primo.

Sustituyendo en (3.3) llegamos al programa final

```
var
   p,n:entero; primo:booleano;
comienzo
   leer (n);
   p:=2;
   mientras  p <= n hacer
        comienzo
            si  n mod p =0  entonces
                comienzo { p divide a n, hay que ver si es primo}
                     primo:= verdadero;
                     i:= 2;
                     mientras  (i < p) ∧ primo  hacer
                           comienzo
                              primo:= (p mod i)     0;
                              i:= i+1
                           fin
                fin
            si primo entonces imprimir (p);
            p:= p+1
        fin
fin
```

3.5 EJERCICIOS

3.3 - Escribir un programa que realice el siguiente cálculo, e imprima el resultado ;

$$\frac{500 + (20 - 8) . 4}{6 - 4}$$

3.4 - Escribir un programa que lea los valores de a , b y c e imprima : a - b.(b - c)

3.5 - Escribir un programa que imprima la suma de los primeros 50 múltiplos de 3

3.6 - Escribir un programa que: a) Lea N
b) Lea N números
c) Imprima su promedio - entero -

3.7 - Escribir un programa que : a) Lea N
b)Lea N números
c) Imprima la suma de los pares y la suma de los impares

3.8 - Escribir un programa para calcular e imprimir los 20 primeros términos de la serie de Fibonacci, definida como :

$$a_1 = a_2 = 1$$
$$a_{m+1} = a_m + a_{m-1}$$

Es decir que cada término es la suma de los dos anteriores. Siendo los dos primeros iguales a 1 .

Fibonacci encontró esta sucesión al tratar un problema relacionado con el crecimiento de una población de conejos.

3.9 - Escribir un programa que imprima los 10.000 primeros números primos.

3.10- Escribir un programa que lea : - n
- x
- $a_0, a_1, a_2, \ldots\ldots, a_n$

e imprima el valor de : $a_0 - a_1.x^1 - a_2.x^2 - a_3.x^3 - \ldots\ldots - a_n.x^n$

3.11- Modificar el programa obtenido en el ejercicio .5 para que no pruebe los números pares.

3.12- Escribir un programa que lea dos números enteros, calcula el máximo común divisor, por el algoritmo de Euclides y lo imprima.

Algoritmo de Euclides MCD (a , b)

Sea a >= b

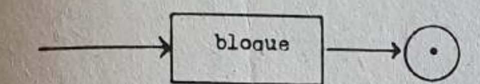
Si se realizan las siguientes operaciones

$$\begin{array}{l|l} a & b \\ \hline r_1 & c_1 \end{array} \quad \begin{array}{l|l} b & r_1 \\ \hline r_2 & c_2 \end{array} \quad \ldots\ldots \quad \begin{array}{l|l} r_{n-1} & r_n \\ \hline 0 & c_n \end{array}$$

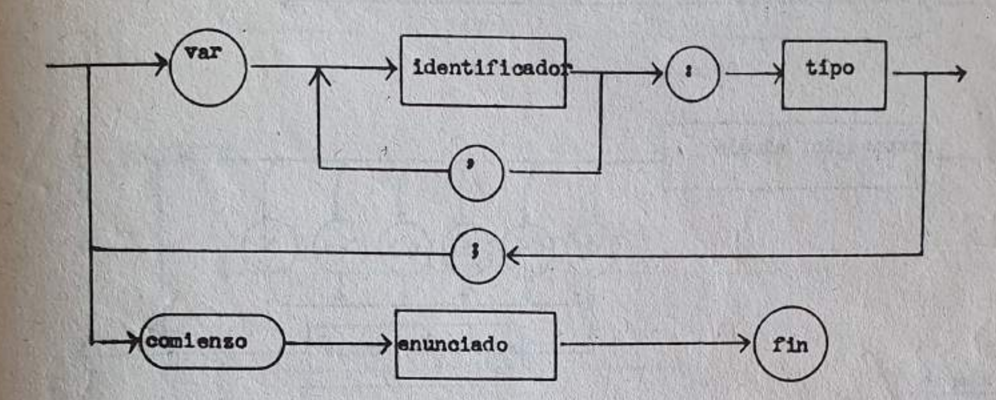
entonces MCD (a ,b) = r_n

3.6 CARTA SINTÁCTICA DE PASCAL-CBC- 0

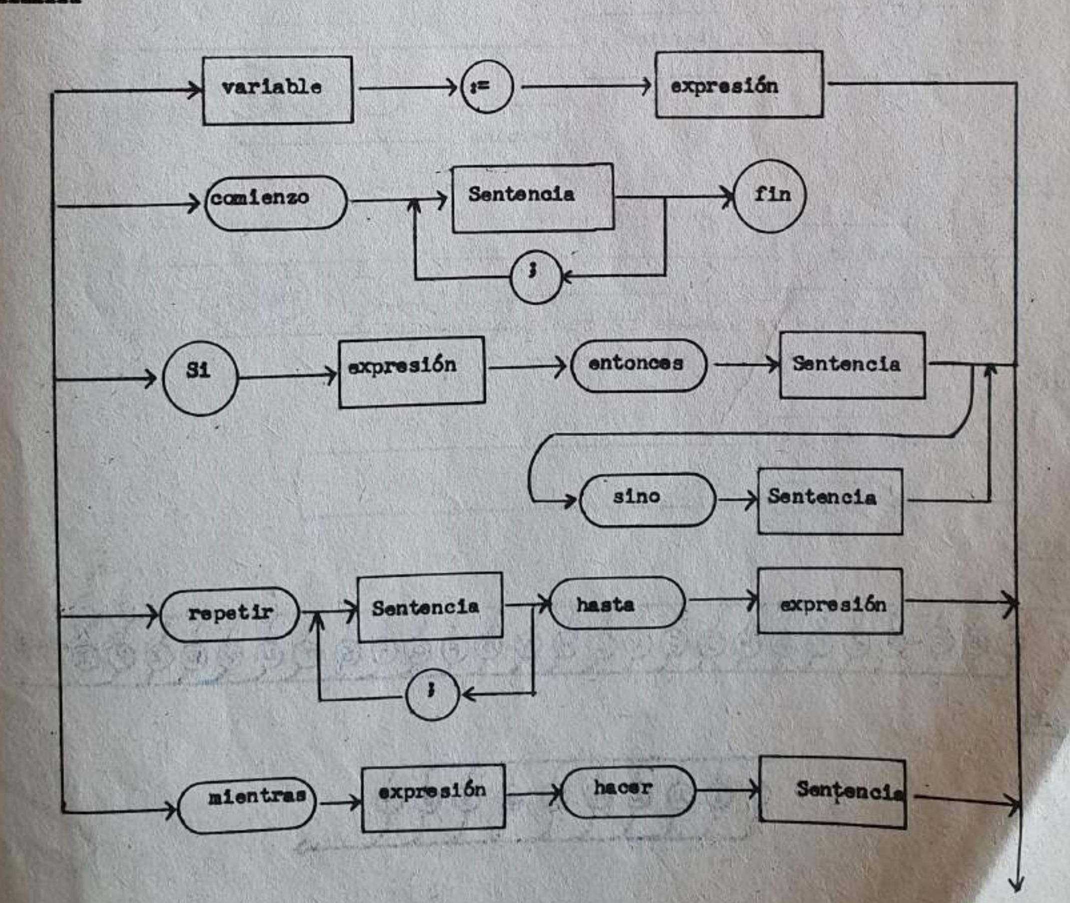
Programa

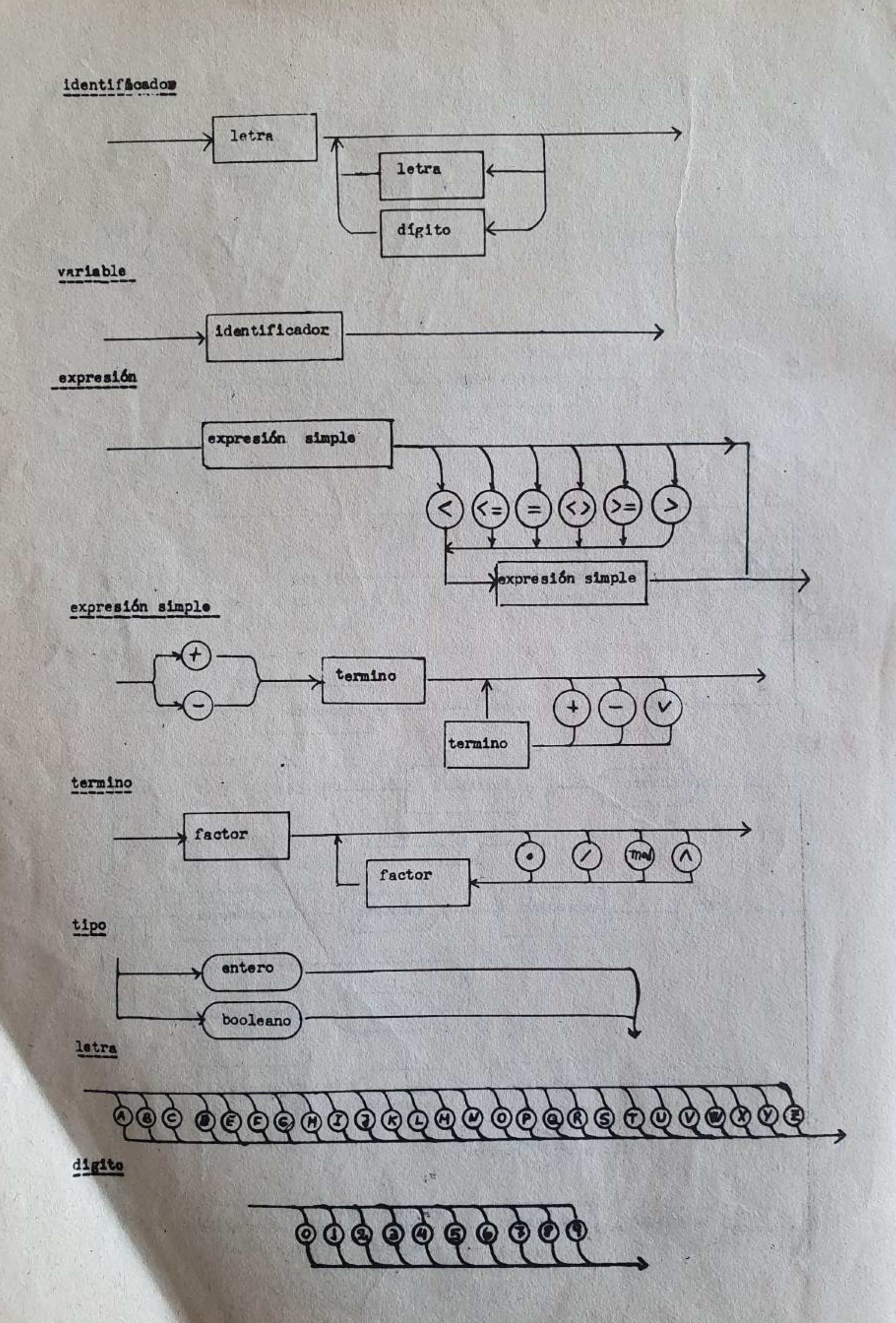


bloque

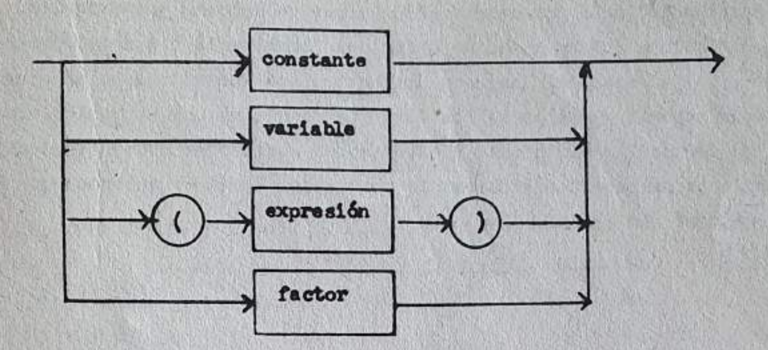


Sentencia

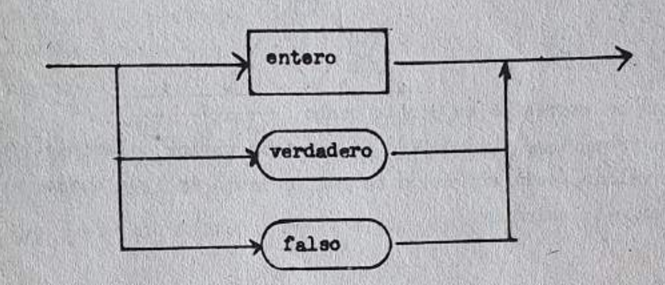



identificador
letra
letra
dígito
variable
identificador
expresión
expresión simple
< <= = <> >= >
expresión simple
expresión simple
+
-
termino
termino
+ - v
termino
factor
factor
· / mod ^
tipo
entero
booleano
letra
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
digito
0 1 2 3 4 5 6 7 8 9

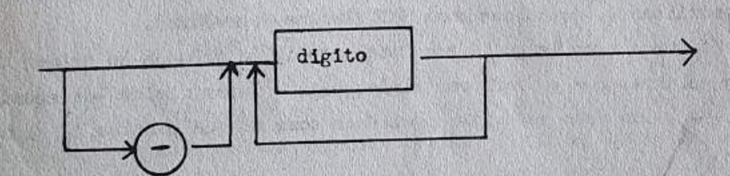
factor



constante

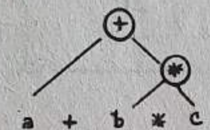


entero

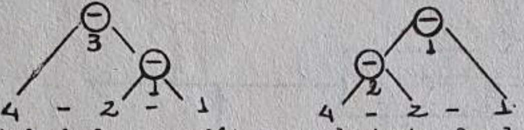


3.7 Anexo : Definición de lenguajes

Todos conocemos ejemplos de la ambiguedad de nuestra lengua, muchas de cuyas frases alcanzan significación solo por enfatización o por relaciones contextuales. Pero aún los mismos lemgua es técnicos que no han sido definidos formalmente o con cuidadoso rigor, son tambien ambiguos, tal el caso del lenguaje aritmético que aprendimos en nuestros cursos elementales. En la mayoría de ellos se nos dice que cuando en una expresión aparecen sumas (o restas) y productos (o divisiones) estos soben realizarse antes que aquellas a menos que la presencia de un paréntesis fuerse lo contrario. Es claro entonces que la expresión a + b*c se evaluará en el orden indicado :



Pero dada la expresión 4 - 2 - 1 cual de las siguientes ordenes de ejecución debemos adoptar ?



El resultado de la expresión es por lo tanto 3 o 1 ?

Para la mayoria de nosotros nada, aprendido en nuestros cursos de aritmética nos dice cual es el ordenamiento válido, esta expresión es por lo tanto ambigua aunque nos inclinemos naturalmente por la segunda solución

3.10.1 Backus - Naur - Form

La necesidad de desterrar toda ambiguedad en la definición de un lenguaje, como la necesidad de crear algoritmos que lo analicen y permitan usarlo en una computadora, llevan a la utilización de mecanismos , conocidos como BNF (Backus-Nour-Form).

Para poder referirnos a un lenguaje con total precisión debemos en primer término diferenciar aquellas palabras que se refieren al lenguaje - metasímbolos -de aquellas que lo integran - símbolos - . Con esto se evitan problemas como el que plantea la siguiente frase castállana :

AGUDA NO ES AGUDA

La que no es una contradicción si, en su primera ocurrencia ' AGUDA ' representa un símbolo o palabra castellana, mientras que en la segunda ocurrencia es un metasímbolo que refiere a palabras castellanas.

En BNF los metasímbolos se denotan orlados con paréntesis angulares '< ',' >' mientras que los símbolos se escriben textualmente asá ' AGUDA ' será un símbolo mientras ' AGUDA ' será un metasímbolo.

Comencemos por analizar un caso muy simple de la gramática castellana . Consideremos las siguientes reglas gramaticales :

1.- <oración> ::= <sujeto> <predicado>
2.- <sujeto> ::= <artículo> <sustantivo>
3.- <predicado> ::= <verbo> <objeto>
4.- <objeto> ::= <artículo> <sustantivo>
5.- <artículo> ::= el
6.- <artículo> ::= la

7.- <verbo> ::= come

8.- <sustantivo> ::= orangután

9.- <sustantivo> ::= banana

Cuyo significado es el siguiente : Para construir oraciones se debe partir del metasimbolo <oración> . En cada paso pueden sustituirse en la secuencia de simbolos construida en el paso anterior, Aquellos metasímbolos que sean parte izquierda de alguna regla por la correspondiente parte derecha.

El procéso termina, produciendo una oración castellana si la secuencia construida no tiene metasímbolos.

Llamemos parte izquierda al símbolo que precede a " ::= " y parte derecha a la cadena o secuencia de símbolos que le siguen. La oración castellana ' el orangután come la banana ' puede obtenerse mediante las siguientes sustituciones :

	regla aplicada
<oración>	
<sujeto> <predicado>	1
<artículo> <sustantivo> <predicado>	2
el <sustantivo> <predicado>	5
el orangután <predicado>	8
el orangután <verbo> <objeto>	3
el orangután come <objeto>	7
el orangután come <artículo> <sustantivo>	4
el orangután come la <sustantivo>	6
el orangután come la banana	9

Que puede sintetizarse en el siguiente gráfico llamado árbol sintactico de la oración

<oración>

<sujeto> <predicado>

<objeto>

<artículo> <sustantivo> <verbo> <artículo> <sustantivo>

el orangután come la banana

Para simplificar la escritura, cuando aparezcan varias reglas con identico miembro izquierdo, como en nuestro caso las reglas 5 y 6 se las puede agrupar en un renglón, separando las distintas partes derachas por una barra vertical, así las reglas 5 y 6 se escribirán :

< artículo> ::= el | la

Si ahora reemplazamos la regla 7.- por : < verbo > ::= come | dicta

y las reglas 8 y 9 por : < sustantivo> ::= orangután | maestro | clase

De esta nueva BNF podremos también derivar las oraciones :

. el maestro dicta clase

. el maestro come la banana

como así tambien

, el orangután dicta clase

o

. la banana dicta clase

Ya que el método simplemente produce oraciones sintacticamente correctas.

Ejercicio a) Construya los árboles sintácticos correspondientes a los ejemplos anteriores.

Ejercicio b) Las siguientes reglas generan representaciones binarias de todos los números naturales .

<número> ::= <cabeza> | 0

<cabeza> ::= <cabeza> 1 | <cabeza> 0 | 1

1º) usando estas reglas derive : 100100

11

2º) se puede derivar 001?

Ejercicio c) Escriba la BNF que genere las representaciones binarias de los números naturales pares.

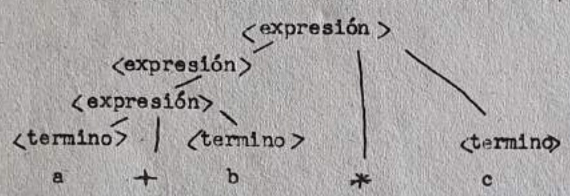
Retornemos al lenguaje en el cual habiamos encontrado una ambiguedad, tratemos de definir su BNF.

<expresión> ::= <expresión> + <termino> | <expresión> * <termino> | <termino> | <expresión>

<termino> ::= <variable>

<variable> ::= a | b | c | d

Construyamos el árbol sintáctico de a + b * c



Es fácil ver que el conjunto de reglas definidas genera todas las expresiones deseadas, sin embargo el árbol sintáctico de este ejemplo tiene una estructura que no es la esperada, ya que la suma se realiza antes que el producto.

En general, en el proceso de interpretación de una sentencia, la aplicación de cada regla sintáctica supone la realización de ciertas acciones o aporta información sobre el sentido de la sentencia, En este caso particular la aplicación de las reglas que involucran a los operadores " + " y " * " supone la realización de las respectivas operaciones.

Vamos a rehacer las reglas de esta BNF de modo que el árbol sintáctico de una expresión manifieste que las multiplicaciones y las divisiones deben realizarse antes que las sumas y las restas (por convención)

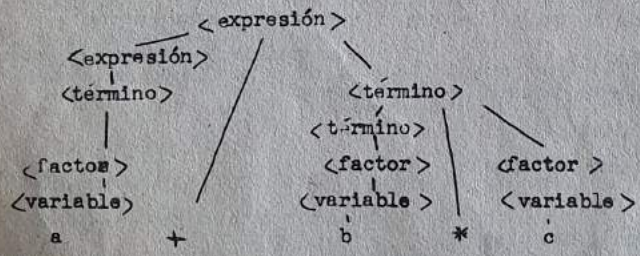
<expresión> ::= <expresión> + <termino> | <expresión> - <termino> | <termino>

<término> ::= <término> * <factor> | <término> / <factor> | <factor>

<factor> ::= <variable> | (<expresión>)

<variable> ::= a | b | c | d

Construyamos ahora el nuevo árbol sintáctico



Ejercicio d) Construya el árbol sintáctico para las siguientes expresiones :

a - b - c

(a + b) * c

3.10.2 Cartas Sintácticas

La BNF no es la única forma de definir un lenguaje, veremos a continuación las cartas sintácticas, debido a su simplicidad y a su amplia difusión

En las cartas sintácticas las reglas se reemplazan por gráficos en los que se usan las siguientes convenciones:

Se utiliza para caracterizar metasímbolos.

Se utiliza para especificar símbolos.

Para explicarar concatenaciones o recursiones.

Para explicitar opciones.

Cada metasímbolo se escribe encabezando el gráfico correspondiente a sus posibles sustituciones ó producciones.

Ejemplos

La carta sintáctica correspondiente al último lenguaje definido es :

expresión

expresión → + → termino
expresión → − → termino
termino

termino

termino → * → factor
termino → / → factor
factor

factor

variable
(→ expresión →)

variable

